Introduction tutorial to YARRRML

Introduction

YARRRML is a Domain Specific Language based on YAML that allows the translation and homogenisation of heterogeneous. Through this language the users can merge data sources in different formats in a single RDF Turtle output, creating a simple script. This avoids the creation of *ad-hoc* solutions for every data source and the consequent data homogeniser.

Structure

Inside a YARRRML script there are some upper categories and some lower categories, as in YAML. Taking this into account, the two mandatory categories will be prefixes and mappings. Inside mappings, we can create different mappings with their own name which will be the root inside the mapping category. Then, inside each mapping we can define a data source, an expression to create the subject and various expressions to create the predicates and objects.

Step by step example

In this example we are going to see how a script can be constructed to merge two data sources (JSON and XML) in a single RDF output. The example will be constructed step by step so that in the end we can have the whole view of the created script.

Data sources

The data sources that we are going to use are two: a set of films in JSON and another set of films in XML. The content of these files can be consulted following the links in Table 1. Each file defines two different films with their corresponding attributes: id, name, year, country and directors. It is important to take into account that inside the Matey editor (web editor for YARRRML) the user must copy and paste the content of these files inside the dedicated text areas in Matey.

Table 1: Enlaces a las fuentes de datos

Format	Link
JSON	https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.json
XML	https://rawgit.com/herminiogg/ShExML/master/src/test/resources/films.xml

Defining prefixes

To define a prefix that will be used in the script and in the RDF output we use the statement showed in Snippet 1. Inside the prefix category different used prefixes are nested. To define a prefix it is sufficient to write the identifier followed by ':' and the URL between double quotes.

```
prefixes:
  ex: "http://example.com/"
```

Snippet 1: Prefix declaration

Defining mappings

To define a mapping the category mappings must be used and then nest inside the name of the new mapping, as it can be seen in Snippet 2.

```
mappings:
  films_json:
```
Snippet 2: Mapping declaration

Defining data sources

Data sources are defined in YARRRML inside a mapping. Therefore, the sources keyword is nested in our previously defined mapping. For the definition of the data source we need three components: the path to the required file, the type of query language to use, and the query defining the iterator. Iterators allow us to define when, inside a file, it is necessary to iterate because more than one entity is present. This is the case of films in the two previously mentioned files. Thus, it will be necessary to define in which part of the file we must iterate and which fields should be taken. This syntax for the YARRRML language can be seen in Snippet 3.

```
mappings:
  films_json:
    sources:
      - ['films.json~jsonpath', '$.films[*]']
```
Snippet 3: Data source declaration

Defining subjects

Subjects allow to define expressions that will be used to generate subjects in the RDF triples. A subject is composed by the prefix of the final IRI plus the partial query in the selected query language of the iterator. Note that to define the partial query it is not necessary to use the navigational operators ("/" en XPath y "." en JSONPath) because the YARRRML engine add them automatically. The whole syntax can be seen in Snippet 4.

```
s: ex:$(id)
```
Snippet 4: Subject declaration

Defining predicates and objects

Predicates and objects allow to define how different RDF triples predicates and objects of an entity will be generated. Each element of this set will be defined by a predicate and an expression that generates the object. The syntax can be seen in Snippet 5. Note that prefixes can be concatenated as with the subjects and that a literal output (by default) or an IRI output can be selected. If we want to change the data type it is possible to add a third argument with the XML Schema data type, like in ex:name.

```
po:
      - [ex:name, $(name), xsd:string]
      - [ex:year, ex:$(year)~iri]
      - [ex:director, $(director)]
      - [ex:country, $(country)]
```
Snippet 5: Predicates and objects declaration

Full example

In the following example we can see how to merge the two previously mentioned files. Note that in this example we can see that because some queries are not equal in both entities it is necessary to create two mappings. In the case that these queries were equal we could use the same mapping and add the both data sources in the sources field.

```
prefixes:
  ex: "http://example.com/"

mappings:
  films_json:
    sources:
      - ['films.json~jsonpath', '$.films[*]']
    s: ex:$(id)
    po:
      - [ex:name, $(name), xsd:string]
      - [ex:year, ex:$(year)~iri]
      - [ex:director, $(director)]
      - [ex:country, $(country)]
  films_xml:
    sources:
      - ['films.xml~xpath', '//film']
    s: ex:$(@id)
    po:
      - [ex:name, $(name), xsd:string]
      - [ex:year, ex:$(year)~iri]
      - [ex:director, $(directors/director)]
      - [ex:country, $(country)]
```

Snippet 6: YARRRML script for films

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ex: <http://example.com/>.

<http://example.com/3> ex:name "Inception";
    ex:year <http://example.com/2010>;
    ex:director "Christopher Nolan";
    ex:country "USA".
<http://example.com/4> ex:name "The Prestige";
    ex:year <http://example.com/2006>;
    ex:director "Christopher Nolan", "Jonathan Nolan";
    ex:country "USA".
<http://example.com/1> ex:name "Dunkirk";
    ex:year <http://example.com/2017>;
    ex:director "Christopher Nolan";
    ex:country "USA".
<http://example.com/2> ex:name "Interstellar";
    ex:year <http://example.com/2014>;
    ex:director "Christopher Nolan", "Jonathan Nolan";
    ex:country "USA".
```

Snippet 7: Turtle result after applying YARRRML script